AI FINAL PROJECT REPORT

Project title: Developing an intelligent agent to play Pong game

Group Members: Ankit Agarwal (MT19021)

Rahul Maheshwari (MT19027)

Problem Statement: Pong is an Atari game whose environment is provided by OpenAI Gym. It is a 2-player game where one of the players is the computer and the other player is the agent which we need to train. Both the players have a paddle which they need to move horizontally or vertically along the screen and hit the ball using it. The opponent scores a point if the player misses the ball. The game gets over when one of the players reaches a score of 21 points. Here, the goal of the agent is to score as many points as possible.

Literature Review: Game playing has been one of the popular topics of research in Artificial Intelligence. The first major breakthrough in this field was the research paper "Playing Atari with Deep Reinforcement Learning" by DeepMind Technologies in the year 2013. This paper uses a modified version of Q-learning along with a Convolution Neural Network where the input to the network are the raw pixels of the game and the output are the Q-values for each of the actions at that state. Also, there has been research done later on in this domain which approaches which further help to train the agent in a faster manner. One of such approaches is given in the paper "Dueling Network Architectures for Deep Reinforcement Learning" by Google DeepMind in 2016.

One further improvement is given in the paper "Human Level Control through deep reinforcement learning". This paper gives the approach of using one more network called Target Network in addition to the main network. The target network is used to calculated the target Q values which are needed while calculating the loss values.

Methodology Followed

Here once the environment is set up the current state of the game is continuously returned in the form of pixel values. The raw Atari frames returned are pixel images with a size of 210 x 160. Directly using them would be too heavy computationally. So, first of all the pre-processing steps are applied

- **Preprocessing Steps:** The frames are first converted form RGB representation to gray-scale values. Further, it is down sampled to a 110 x 84 image and also some of the top and below portion of the game screen is not useful and does not give any useful information about the current state of the game. Therefor it is also cropped and the image is reduced to a size of 84 x 84.
- **Exploration-exploitation policy:** This is a common policy which is used while applying reinforcement learning. We want the agent to do more of exploration in the beginning and less of exploitation because it has almost no knowledge of the environment. The reverse of this approach needs to be followed as the game progresses. This is done by appropriately choosing epsilon decay values and minimum epsilon values.
- **Storing the state of the game:** It is not possible to know the direction of the movement of the ball in the game by just looking at one frame. So, here the state of the game is represented by stacking 4 continuous frames together and then storing them.
- **Creating Replay Memory:** Here a replay memory is needed so that while the network being trained then the samples on which it is being trained are not just consecutive samples and can be randomly drawn from the memory. Transitions are stored in the replay memory where each transition consists of [state, action, reward, terminal, new_state] which is a list. A state as discussed previously is formed by stacking 4 consecutive frames and the new_state is

formed by replacing the oldest frame in the state by the new frame. The variable reward denotes the reward that the agent gets on going from state to new_state and terminal denotes whether the episode terminates on reaching new_state.

- **Q learning:** The main network predicts the Q value by taking a batch of 32 transitions from the replay buffer that is being created continuously. The Q value predicted by the network is called the Qpredicted. Now the loss function is calculated using the Qpredicted value and the Qtarget value obtained using the Bellman's equation. One approach would be to calculate the Qtarget value using the main network only and thus using the parameters which are continuously changing. A better approach is to use a separate network called target network for calculating Qtarget. We have used the approach of using a separate target network.
- **Target network:** The target network as discussed above is used to find Qtarget values. The weights of this network are not being updated continuously. The main network weights are copied to the target network only after specific number of steps.
- **Loss function:** We have used the Huber loss function for calculating the loss between Qpredicted and Qtarget. The model is made to fit using these loss values i.e. the weights are updated.
- **Double Q Learning:** As our agent was not playing well after implementing all the above concepts, we then also applied Double Q learning methodology. The method has a slight different approach for the Q-value of the next state that is used in the Bellman equation. The difference in the Bellman equation used to calculate the Qtarget value in the 2 approaches is shown below:

  Normal Q-learning-
  $Qtarget(s, a) = r + \gamma max(s', a'; \theta target)$

  Double Q-learning-
  $Qtarget(s, a) = r + \gamma Q(s', a' = argmaxQ(s', a'; \theta main); \theta target)$

  In Double-Q learning, which action is best in the next state i.e. s' is estimated by the main network and then the Q' value for that action is predicted by the target network. Now this value is used is used in the Bellman equation to find the Qtarget value for the state. This approach solves the problem of over-estimation in Q values due to noise in the network. As the calculation of Q' value has direct involvement of 2 networks and the 2 networks will have different noise therefore the bias that may be there in the Q' value eventually cancels out.
- **Network structure:** The network has 4 convolution layers with different filter sizes. The activation function used is relu. There is one fully connected final layer.

Results

We have tried varying the parameter values used in the model but still the agent is not able to play well the reason being the hardware constraints. Here we are able to train the agent for more than 300 episodes as the RAM requirement shoots up drastically.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller – "Playing Atari with Deep Reinforcement Learning"

[2] Ziyu Wang, Tom Schaul, Matteo Hessel, M Hado van Hasselt, Marc Lanctot, Nando de Freitas – "Dueling Network Architectures for Deep Reinforcement Learning"

[3] Mnih et al. 2015 – "Human-level control through deep reinforcement learning"